Nombre del proyecto
Automatización de facturas vehiculares

Profesor	Mgt. Gladys María Villegas Rugel
Materia	PROCESAMIENTO DE LENGUAJE NATURAL
Alumnos	Andrea Fernanda Morán Vargas Pedro José Vidal Orús
Fecha	24 de septiembre 2025

Resumen del Proyecto

Análisis Comparativo de Algoritmos de IA

Objetivo
Realizar una investigación y comparación exhaustiva de mínimo 5 técnicas de Inteligencia Artificial aplicables al problema definido en su proyecto, evaluando pros/contras, complejidad computacional y casos de uso típicos.

Técnicas Mínimas a Evaluar
Debe incluir algunas de las siguientes 5 técnicas:

- Random Forest
- Support Vector Machines (SVM)
- Neural Networks (Redes Neuronales)
- Transformers
- Una quinta técnica relevante a su proyecto (ej: XGBoost, K-Means, LSTM, CNN, etc.)

Tabla de Contenido

Comparativa de técnicas de IA para la extracción de información de facturas vehiculares 3
Random Forest 3
1. Descripción Teórica 3
2. Ventajas y Desventajas 3
Desventajas: 4
3. Complejidad Computacional 4
4. Casos de Uso Típicos 5
5. Aplicabilidad a su Proyecto 6
Support Vector Machines (SVM) 7
1. Descripción Teórica 7
2. Ventajas y Desventajas 7
Desventajas: 8
3. Complejidad Computacional 8
4. Casos de Uso Típicos 9
5. Aplicabilidad a su Proyecto 10
Redes Neuronales (Artificial Neural Networks) 11
1. Descripción Teórica 11
2. Ventajas y Desventajas 12
3. Complejidad Computacional 13
4. Casos de Uso Típicos 14
5. Aplicabilidad a su Proyecto 15
Transformers 16
1. Descripción Teórica 16
2. Ventajas y Desventajas 17
3. Complejidad Computacional 19
4. Casos de Uso Típicos 20
5. Aplicabilidad a su Proyecto 21
Redes Neuronales Convolucionales (CNN) 23
1. Descripción Teórica 23
2. Ventajas y Desventajas 24
3. Complejidad Computacional 25
4. Casos de Uso Típicos 26
5. Aplicabilidad a su Proyecto 26

Comparativa de técnicas de IA para la extracción de información de facturas vehiculares

Random Forest

1. Descripción Teórica

Un Random Forest es un algoritmo de aprendizaje supervisado de tipo ensemble basado en árboles de decisión. Combina múltiples árboles de decisión entrenados sobre diferentes subconjuntos de datos y usando subconjuntos aleatorios de características en cada división, para producir un modelo más robusto. Cada árbol vota por una clase o predice un valor, y el bosque agrega estos resultados para obtener la predicción final. Este enfoque reduce el sesgo de modelos simples a la vez que controla la varianza al promediar muchos estimadores débiles. Los fundamentos matemáticos se basan en los algoritmos de árbol de decisión (por ejemplo, CART, que usa criterios como Gini o entropía para divisiones) y en la teoría de bagging.

Los hiperparámetros principales de Random Forest incluyen: el número de árboles del bosque, la máxima profundidad de cada árbol, y el número de características candidatas consideradas en cada nodo para buscar la mejor división. Al ser un método no paramétrico, no asume distribuciones específicas en los datos y puede manejar tanto variables numéricas como categóricas.

2. Ventajas y Desventajas

Ventajas:

- **Menor sobreajuste:** al promediar múltiples árboles se reduce el riesgo de sobreajustar el modelo; un bosque aleatorio con suficientes árboles tiende a no sobreajustar, ya que el promedio de árboles no correlacionados disminuye la varianza global.
- **Flexibilidad y precisión:** puede usarse tanto para clasificación como para regresión con alta precisión, manejando bien datos de alta dimensionalidad y mezclas de características numéricas/categóricas. No requiere que los datos estén normalizados ni transformados, ya que es invariante a escalas y combinaciones monotónicas de las variables.
- **Robustez a datos faltantes y outliers:** utiliza bagging de características, por lo que puede mantener desempeño aun si faltan algunas variables; asimismo, los árboles individuales son poco

sensibles a valores atípicos extremos en los predictores.

Desventajas:

- **Menor interpretabilidad:** al combinar muchos árboles, el modelo resultante se vuelve más difícil de interpretar que un solo árbol de decisión. Ya no es trivial seguir la "ruta de decisión" como en un árbol único, lo que complica explicar las predicciones.
- **Mayor costo computacional:** entrenar y almacenar múltiples árboles aumenta el uso de cómputo y memoria. Random Forest puede ser lento en problemas con muchísimos árboles o muchos atributos, ya que para cada árbol se realizan búsquedas de splits en varias variables. De igual forma, la fase de predicción requiere recorrer todos los árboles, lo cual puede ser más lento que modelos más simples.
- **Bias en datos con muchas categorías:** el proceso aleatorio de división puede favorecer, por azar, a variables con muchos niveles o continuas al encontrar puntos de corte óptimos. Esto puede llevar a que ciertas variables aparezcan artificialmente más importantes. Además, los árboles categorizan variables continuas en intervalos; esta discretización puede conllevar pérdida de información.

3. Complejidad Computacional

La complejidad temporal de entrenar un Random Forest es aproximadamente proporcional al número de árboles T multiplicado por la complejidad de entrenar cada árbol. Entrenar un solo árbol de decisión mediante algoritmos como CART tiene un costo promedio del orden de $O(n \cdot m\log n)$ (siendo n el número de muestras y m el número de características, asumiendo árboles balanceados). Para el bosque completo, la complejidad se escala casi linealmente con T . Más formalmente, una estimación común es: $O(T \cdot n \cdot k \cdot \log n)$, donde k es el número de características consideradas en cada división (si k = m, sería $O(T \cdot n \cdot m\log n)$). En Random Forest típicamente k es un subconjunto menor que m (p. ej., $\sqrt{m}$ en clasificación), lo que acelera cada split.

En cuanto a la complejidad de predicción, para clasificar una nueva instancia, ésta debe recorrer cada uno de los T árboles. Cada recorrido de árbol tiene un costo proporcional a su profundidad D (número de divisiones desde la raíz a una hoja). Si los árboles están equilibrados, $D \approx \log_2(n)$. Así, la predicción para un ejemplo nuevo cuesta en torno a $O(T \cdot D)$, que típicamente es $O(T\log n)$ por instancia. En la práctica, con árboles poco profundos o limitados, la inferencia es bastante rápida, aunque con muchos árboles puede ser más lenta que la de modelos lineales.

La complejidad espacial reside principalmente en almacenar los T árboles. Cada árbol puede ocupar memoria proporcional al número de nodos que contiene; en el peor caso, un árbol sin podar podría tener del orden de $O(n)$ nodos (si se divide casi hasta hojas puras). Random Forest usualmente limita la profundidad máxima o el tamaño mínimo de hojas para controlar esto. En conjunto, el modelo podría tener del orden de $O(T \cdot n)$ nodos en el peor caso, aunque normalmente es bastante menor. Adicionalmente, almacenar algunos resultados out-of-bag y cálculos de importancia añade un sobrecoste mínimo. En términos de escalabilidad, Random Forest maneja bien conjuntos de datos grandes gracias a su naturaleza paralelizabla (cada árbol se entrena independiente). Su entrenamiento escala casi linealmente con n y puede distribuirse en múltiples núcleos o máquinas. Sin embargo, a medida que n y m crecen mucho, el tiempo de entrenamiento y la memoria aumentan correspondientemente, en particular, métodos exactos de búsqueda de splits tienen dificultad con m muy grande, aunque existen variantes aproximadas para alta dimensionalidad.

4. Casos de Uso Típicos

Random Forest es una técnica muy versátil y se ha aplicado con éxito en numerosos dominios:

- **Finanzas y banca:** ampliamente usado en credit scoring y detección de fraudes. Por ejemplo, puede predecir la probabilidad de impago de un préstamo o identificar transacciones bancarias fraudulentas, aprovechando su capacidad para manejar muchas variables financieras y detectar interacciones complejas. También se emplea en seguros para modelar riesgo y fijación de primas.

- **Medicina y salud:** en tareas de diagnóstico médico asistido, Random Forest ha sido utilizado para clasificar imágenes médicas y para predecir resultados clínicos a partir de datos de pacientes. Su robustez a entradas ruidosas y relevancia de variables resulta útil en bioinformática, por ejemplo, para seleccionar genes o biomarcadores relacionados con cierta enfermedad.

- **Industria y manufactura:** en mantenimiento predictivo, para predecir fallos de maquinaria a partir de sensores, Random Forest funciona bien dada su tolerancia a datos ruidosos y correlacionados. También en control de calidad, para clasificar productos defectuosos analizando múltiples mediciones.

- **Marketing y retail:** aplicado a segmentación de clientes y predicción de comportamiento. Por ejemplo, puede modelar la tasa de abandono de clientes (churn) o recomendar productos, analizando numerosas variables demográficas y de historial de compras. Empresas de

comercio electrónico los usan para predecir qué usuarios abandonarán el carrito de compra y así tomar acciones preventivas.

5. Aplicabilidad a su Proyecto

El problema propuesto consiste en extraer información estructurada de facturas vehiculares (documentos) a partir de imágenes u PDF escaneados, combinando OCR e IA. En este contexto, Random Forest podría intervenir en ciertas etapas específicas, aunque no es la técnica más típica para el núcleo del reconocimiento de texto (donde generalmente se usan redes neuronales profundas). Algunas posibles aplicaciones dentro del proyecto serían:

- **Clasificación post-OCR:** Tras extraer texto de la factura con OCR, un Random Forest podría usarse para clasificar el documento o partes del mismo. Por ejemplo, distinguir si una factura pertenece a cierto formato o provincia (según su contenido), o detectar si el documento escaneado es legible o necesita reescanearse.

- **Detección de anomalías o validación:** Otra posible aplicación es verificar la coherencia de los datos extraídos. Por ejemplo, se podrían entrenar modelos Random Forest para predecir un campo numérico (como el total de la factura) a partir de otros campos (impuestos, subtotales) y así comprobar discrepancias entre lo predicho y lo leído por el OCR, señalando posibles errores de extracción.

- **No apto para reconocimiento de texto en imágenes:** Es importante notar que Random Forest no es adecuado para la etapa de lectura de caracteres o palabras en la imagen. Las tareas de OCR moderno requieren analizar patrones visuales complejos (formas de letras, etc.) donde métodos basados en árboles no son competentes en comparación con redes neuronales convolucionales o modelos de aprendizaje profundo entrenados directamente sobre pixeles. Por tanto, no se usaría Random Forest para reconocer el texto en la factura, sino más bien complementariamente para procesos de clasificación o verificación posteriores sobre los datos ya extraídos.

- **Viabilidad:** Implementar Random Forest en el proyecto sería relativamente sencillo utilizando librerías estándar (como scikit-learn). Requiere preparar vectores de características numéricas a partir del texto extraído (por ejemplo: frecuencias de ciertas palabras, longitudes de campos, etc.). Dado que suele haber abundantes datos históricos de facturas, entrenar un Random Forest no demandaría recursos extremos – una CPU normal podría manejar cientos de árboles en segundos. Se debe cuidar, eso sí, no generar árboles demasiado profundos que modelen ruido presente en datos OCR (que pueden contener errores).

Random Forest podría incluirse en el proyecto principalmente para clasificar o validar información extraída más que para la extracción en sí. Si bien no es la piedra angular del OCR, aporta una herramienta poderosa y fácil de usar para tareas auxiliares de decisión basada en datos estructurados derivados del documento

Support Vector Machines (SVM)

1. Descripción Teórica

Las Máquinas de Vectores de Soporte (SVM) son algoritmos de aprendizaje supervisado usados principalmente para clasificación (y también para regresión, en su variante SVR). Teóricamente, una SVM busca encontrar el hiperplano óptimo que separe las clases en un espacio de características de dimensión N de forma que el margen (la distancia entre dicho hiperplano y los puntos de datos más cercanos de cada clase) sea máximo. Este enfoque de máximo margen garantiza una buena generalización de la clasificación a nuevos datos. Matemáticamente, en su forma lineal básica, se resuelve un problema de optimización cuadrática para maximizar ese margen sujeto a que todos los puntos estén del lado correcto del hiperplano.

Una de las claves de las SVM es que pueden realizar clasificación no lineal mediante el truco del kernel. Si los datos no son separables linealmente en el espacio original, se utilizan funciones kernel para proyectarlos implícitamente a un espacio de dimensión mayor donde sí sean separables. Comúnmente se emplean kernels como el radial (RBF gaussiano), polinomial o sigmoide, entre otros. Estos kernels permiten calcular productos internos en el espacio proyectado sin tener que computar coordenadas en ese espacio de alta dimensión, lo cual hace tractable la optimización.

El algoritmo de entrenamiento identifica un subconjunto de ejemplos llamados vectores de soporte, que son aquellos puntos de entrenamiento que "tocan" los límites del margen óptimo. Solo estos vectores de soporte influyen en la definición del hiperplano final; puntos más alejados no afectan la solución. Esto hace que la solución de la SVM esté definida por unos pocos ejemplos críticos.

2. Ventajas y Desventajas

Ventajas:

- Efectivo en alta dimensión: Las SVM son muy eficaces cuando el número de características m es grande, incluso mayor que el número de muestras n. Manejan bien datos de alta dimensionalidad gracias a que el optimizador se enfoca en vectores de soporte y no necesita hacer un modelado explícito de cada variable.

- **Uso eficiente de ejemplos relevantes:** Solo una fracción de los datos (los vectores de soporte) determinan el modelo final, lo que las hace eficientes en memoria y en cómputo de predicción, ya que no necesitan evaluar todos los puntos de entrenamiento, sino únicamente los soportes.
- **Modelo robusto y con buen poder de generalización:** Al maximizar el margen, las SVM suelen lograr alta capacidad de generalización, reduciendo el riesgo de sobreajuste. En escenarios bien condicionados (con datos separables aproximadamente), la frontera de máximo margen suele resultar en clasificadores que funcionan muy bien con datos nuevos. Además, incorporan de forma natural cierta resistencia a outliers mediante el margen blando.

Desventajas:

- **Escalabilidad pobre con muchos datos:** El principal inconveniente práctico de las SVM es su complejidad de entrenamiento. Entrenar una SVM requiere resolver un problema de optimización cuadrática cuyo costo crece entre $O(n^2)$ y $O(n^3)$ (dependiendo de cómo se aproveche la estructura y de la eficacia de la caché en el solver). En conjuntos de datos muy grandes (decenas de miles de ejemplos o más), el entrenamiento puede volverse muy lento o directamente inviable en términos de memoria, ya que involucra manejar una matriz kernel de tamaño $n \times n$.
- **Dificultad de ajuste de hiperparámetros:** A diferencia de métodos más automatizados, las SVM requieren ajustar con cuidado parámetros como el tipo de kernel, C, γ , etc., normalmente mediante validación cruzada. Un kernel inapropiado o mal parametrizado puede dar resultados pobres.
- **No proporcionan probabilidades directas:** Las SVM originales producen una decisión determinista (clasificación dura). Obtener **estimaciones de probabilidad** requiere un paso adicional de calibración (por ejemplo, usando Platt scaling con validación cruzada), lo cual añade coste computacional.

3. Complejidad Computacional

El entrenamiento de una SVM implica resolver un problema de optimización cuadrática con n variables (en el caso de clasificación binaria estándar). El algoritmo clásico (como el utilizado en LIBSVM) tiene una complejidad temporal teórica que escala aproximadamente entre $O(n^2 \cdot m)$ y $O(n^3 \cdot m)$ en función de la eficacia con que se explote la estructura del kernel y la utilización de la caché durante la optimización. En la práctica, esto significa que el tiempo de entrenamiento crece cuadráticamente con el número de ejemplos en muchos casos, volviendo prohibitivos los conjuntos de entrenamiento muy grandes. Para SVM lineales existen algoritmos más eficientes (por ejemplo, la implementación en LIBLINEAR) que pueden entrenar en tiempo casi lineal con respecto a n y m, escalando a millones de muestras mediante técnicas de descenso de gradiente estocástico. Sin

embargo, para SVM con kernel no lineal, entrenar sobre más de algunas decenas de miles de puntos suele ser inviable sin recurrir a métodos aproximados.

En términos de predicción, como se mencionó, se requiere calcular la función de decisión sumando las contribuciones de cada vector de soporte. Si denotamos $s = |SV|$ el número de vectores de soporte retenidos tras el entrenamiento, y m el número de características (o la dimensionalidad implícita del espacio del kernel, aunque para el cálculo práctico se usan dot products), la complejidad por instancia nueva es $O(s \cdot m)$. En el mejor de los casos, $s \ll n$ (pocos vectores de soporte), por lo que la predicción es rápida. Pero en el peor caso s puede ser del mismo orden que n. Esto implica que, en problemas donde la SVM necesite muchos soportes, clasificar nuevos ejemplos puede ser relativamente lento comparado con modelos lineales simples (que serían $O(m)$). Afortunadamente, en muchos problemas reales s tiende a ser moderado. Además, para SVM con kernel RBF, el costo por ejemplo es $O(s \cdot m)$ donde m en realidad es el costo de evaluar la similitud RBF con un soporte (también proporcional a la dimensión original). Es decir, si hay 1000 vectores de soporte y 100 características, se harían 1000 productos internos de 100 dimensiones para clasificar un ejemplo: 100k operaciones.

La complejidad espacial del entrenamiento es significativa: almacenar la matriz kernel completa requiere $O(n^2)$ memoria. Durante el entrenamiento, los algoritmos tipo SMO almacenan parámetros duales para cada ejemplo, lo que también es $O(n)$. Tras el entrenamiento, el modelo debe retener los vectores de soporte, sus etiquetas y coeficientes asociados, además de cualquier parámetro del kernel. Esto suele ser menos costoso que almacenar todo el conjunto (ya que $s < n$ generalmente), pero si s es grande, el modelo puede ocupar un tamaño comparable al conjunto de datos original. Por ejemplo, si se conservan 5000 vectores de soporte de dimensión 300, el modelo almacena 1.5 millones de parámetros aproximadamente (más bias y otros metadatos).

En cuanto a escalabilidad, las SVM no son la opción más escalable para big data. Para grandes n, existen aproximaciones como Linear SVM (con SGD) para casos lineales, o métodos de kernel aproximado para reducir complejidad. Pero la SVM clásica con kernel completo se limita típicamente a unos pocos miles o decenas de miles de ejemplos con hardware convencional. Su fortaleza está más bien en escenarios de tamaño moderado pero alta dimensión, donde otros modelos podrían tener dificultad.

4. Casos de Uso Típicos

Las SVM tuvieron gran auge en las décadas de 1990 y 2000 y se aplicaron en un amplio rango de tareas de clasificación y regresión. Algunos **casos de uso típicos** son:

- **Clasificación de texto (NLP):** Antes del advenimiento de las redes neuronales profundas, las SVM eran un estándar en tareas de procesamiento de lenguaje natural. Se usaron para análisis de sentimientos, detección de spam, clasificación de noticias o de documentos por temas, etc.

- **Sistemas de información geográfica (GIS):** En geociencia, las SVM se han adoptado para analizar datos geoespaciales. Un caso es la clasificación de cobertura terrestre con imágenes satelitales: dado el espectro de cada píxel, una SVM puede clasificar si es agua, bosque, cultivo, urbano, etc.

5. Aplicabilidad a su Proyecto

En el contexto de extraer información estructurada de facturas vehiculares con OCR, las SVM podrían desempeñar un rol complementario, aunque no central, en el pipeline de IA:

- **Clasificación de segmentos de texto:** Dada una factura escaneada, tras realizar OCR es posible que necesitemos identificar qué partes del texto corresponden a ciertos campos (por ejemplo, distinguir el número de placa del número de chasis, del nombre del propietario, etc.). Una estrategia podría ser extraer características de cada campo reconocido (como palabras clave cercanas, posición en el documento, longitud del texto, formato de caracteres) y entrenar una SVM para clasificar fragmentos de texto en categorías predefinidas (placa, VIN, fecha, monto, etc).

- **Verificación de autenticidad o detección de fraudes:** Si el proyecto Verifactura incluyera verificar si una factura es legítima o potencialmente fraudulenta, podríamos modelarlo como un problema de clasificación binaria donde la entrada son características extraídas del documento (inconsistencias, firmas, logos, patrones inusuales).

- **Limitaciones y viabilidad:** Hay que tener en cuenta que las SVM no son idóneas para el procesamiento directo de imágenes crudas de las facturas. A diferencia de CNNs, una SVM no puede ingerir píxeles sin una etapa previa de extracción de características manual. Por tanto, no usaríamos SVM para reemplazar a un motor OCR basado en visión por computador.

- **Justificación de uso o exclusión:** Las Máquinas de Vectores de Soporte (SVM) no resultan idóneas para el flujo principal de Verifactura porque no pueden procesar imágenes crudas de facturas, a diferencia de arquitecturas como CNNs o Transformers, y requieren etapas manuales de extracción de características que reducen la automatización del sistema. Además, su entrenamiento presenta una

complejidad computacional elevada que escala mal con datasets grandes y de alta dimensionalidad, lo que las hace poco prácticas en escenarios donde existe gran diversidad de formatos y ruido derivado del OCR.

Redes Neuronales (Artificial Neural Networks)

1. Descripción Teórica

Las Redes Neuronales Artificiales (RNA) son modelos de aprendizaje automático inspirados libremente en el funcionamiento del cerebro humano. En esencia, una red neuronal es un conjunto de unidades de procesamiento interconectadas (nodos o neuronas artificiales) organizadas en capas, donde cada neurona calcula una combinación ponderada de sus entradas, aplica una función no lineal (función de activación) y propaga el resultado a la siguiente capa. Una red neuronal típica consta de: una capa de entrada, una o varias capas ocultas internas donde ocurre la mayor parte del cómputo y aprendizaje de patrones, y una capa de salida que produce la predicción final.

Matemáticamente, cada neurona realiza una operación del tipo: $z = f(\sum_i w_i\, x_i + b)$, donde x_i son sus entradas (salidas de neuronas de la capa previa o inputs originales), w_i son pesos sinápticos ajustables, b es un sesgo, y f es la función de activación (por ejemplo sigmoide, ReLU, tanh). El aprendizaje de la red consiste en encontrar los valores de los pesos w (y sesgos) que minimizan un cierto error en las predicciones. Esto se logra típicamente mediante propagación hacia atrás (backpropagation), un algoritmo que calcula el gradiente del error respecto a cada peso propagando el error desde la salida hasta la entrada, y ajusta los pesos usando descenso por gradiente.

Existen diversos tipos de aprendizaje con redes neuronales: la gran mayoría de aplicaciones utilizan aprendizaje supervisado, donde se proveen ejemplos de entrada con sus salidas deseadas para entrenar la red (p. ej., imágenes etiquetadas con clases). También hay redes neuronales usadas en aprendizaje no supervisado (como autoencoders que aprenden a comprimir datos sin etiquetas) e incluso aprendizaje por refuerzo (redes que aprenden estrategias mediante recompensas). Pero las arquitecturas básicas – perceptrones multicapa, CNN, RNN, transformers – suelen entrenarse con un objetivo supervisado o auto-supervisado.

Los hiperparámetros principales de una red neuronal incluyen su arquitectura (número de capas y neuronas por capa), la función de activación en cada capa (ReLU, sigmoide, etc.), el algoritmo de optimización (p. ej., SGD, Adam) y la tasa de aprendizaje, número de épocas de entrenamiento, tamaño de batch, etc. Todos estos afectan cómo y qué aprende la red. Las redes neuronales profundas

(deep learning) simplemente refieren a redes con muchas capas ocultas (más de dos), que pueden modelar funciones sumamente complejas a costa de requerir mucho más datos y cómputo para entrenar.

2. Ventajas y Desventajas

Ventajas:

- **Capacidad de modelar patrones complejos:** Las redes neuronales, especialmente las profundas, pueden aprender y reconocer patrones altamente no lineales y complejos que otros algoritmos no capturan. Por ejemplo, pueden inferir características latentes en imágenes (como conceptos abstractos) o relaciones sutiles en datos de series temporales.
- **Aprendizaje automático de características:** A diferencia de métodos tradicionales que requerían mucha ingeniería de atributos manual, las redes profundas aprenden representaciones intermedias de los datos por sí solas. Cada capa oculta descubre una transformación útil de los datos para la tarea final. Esto reduce la necesidad de conocimiento experto del dominio para crear features, la red extrae de forma autónoma las características relevantes.
- **Versatilidad y amplio rango de aplicaciones:** Las redes neuronales se han aplicado exitosamente en multitud de campos: visión por computadora, procesamiento de lenguaje natural, robótica, predicción en finanzas, juegos, sistemas de recomendación, etc. Soportan datos de diversa naturaleza (imágenes, audio, texto, series) con adaptaciones arquitectónicas (CNN para imágenes, RNN/Transformers para secuencias, etc.).

Desventajas:

- **Necesidad de grandes volúmenes de datos**: Las redes neuronales típicamente requieren muchos datos de entrenamiento para ajustar millones de parámetros sin sobreajustar. Esto puede ser un desafío en áreas donde recolectar datos es costoso o los eventos de interés son raros.
- **Altos requerimientos computacionales:** El entrenamiento de redes profundas es computacionalmente costoso. Implica realizar gran cantidad de operaciones de álgebra lineal (matriz-vector) y almacenamiento de gradientes. Se suelen necesitar GPUs u otro hardware especializado para que el entrenamiento tome horas o días en vez de meses.
- **Opacidad e interpretabilidad limitada:** Las redes neuronales son proverbialmente "cajas negras". Aunque existen métodos de interpretabilidad (visualización de activaciones, feature importance, etc.), en general es difícil explicar por qué la red tomó una decisión particular.

3. Complejidad Computacional

El costo computacional de las redes neuronales depende en gran medida de la arquitectura (número de capas y neuronas) y del tamaño del conjunto de datos. Podemos desglosarlo en entrenamiento e inferencia:

- **Entrenamiento:** Cada iteración de entrenamiento (una pasada por un batch de datos) consiste en un paso hacia adelante (forward) calculando activaciones capa por capa, y un paso hacia atrás (backpropagation) calculando gradientes. Si consideramos una red feed-forward densa simple con L capas, donde la capa l tiene n_l neuronas y n_{l-1} entradas de la capa previa, entonces la complejidad para el forward de esa capa es $O(n_{l-1} \cdot n_l)$ (multiplicar la entrada por la matriz de pesos de tamaño $n_{l-1} \times n_l$) Sumando sobre todas las capas, la complejidad de un forward-pass es $O\left(\sum_{l=1}^{L} n_{l-1} \cdot n_l\right)$. El backward-pass tiene costo similar, proporcional al número de pesos también, ya que calcula gradientes para cada parámetro en la red. Por tanto, el costo por ejemplo (o por batch) es proporcional al número total de pesos en la red. Si P es el número de parámetros (pesos y bias), el forward+backward para un ejemplo es $O(P)$. Para N ejemplos en un epoch completo, es $O(N \cdot P)$. Formalmente, se puede expresar que una red con capas $i \to j \to k \to l$ (por ejemplo 4 capas) tiene complejidad $O\big(N \cdot (i \cdot j + j \cdot k + k \cdot l)\big) = O(N \cdot P)$ por época. Esto suele ser bastante grande pero altamente paralelizable: las operaciones son sumas y multiplicaciones que pueden distribuirse en muchos núcleos o aceleradores. En la práctica, entrenar redes grandes es intensivo pero aprovechando GPUs se logra en tiempos razonables. La complejidad espacial durante entrenamiento es también elevada: se deben almacenar los pesos P y además las activaciones intermedias y gradientes de cada neurona para backpropagation. Esto puede duplicar o triplicar la memoria respecto a solo almacenar el modelo. En redes profundas, la memoria de GPU puede ser el factor limitante por la necesidad de guardar activaciones de miles/millones de neuronas para calcular gradientes.

- **Inferencia:** Para la predicción, solo se realiza el paso forward. Así, clasificar una instancia nueva tiene un costo de $O(P)$ operaciones (donde P es el número de conexiones/pesos activos, que es similar al calculado anteriormente). Por ejemplo, en una red densa con 1000 neuronas en la primera capa y 100 en la segunda, serían 1000×100 ops en la primera + etc. Las redes convolucionales tienen costos distintos por la naturaleza de las convoluciones, pero igualmente escalables con el tamaño de la entrada y filtros. Dado que tras entrenamiento P es fijo, para usos en producción las redes neuronales pueden procesar múltiples muestras en paralelo muy eficientemente usando vectores/matrices en GPUs. Por eso se logran sistemas de

inferencia en tiempo real (ej: detectar objetos en video 30 FPS) con modelos bien optimizados. La complejidad espacial de almacenar el modelo para inferencia es básicamente guardar los pesos (que pueden ser desde unos KB para redes pequeñas, hasta cientos de MB para redes muy grandes tipo GPT-3 con billones de parámetros). En dispositivos con recursos limitados, a veces hay que reducir modelos (podar neuronas, cuantizar pesos) para caber en memoria.

En cuanto a escalabilidad, las redes neuronales entrenan más lento cuanto más datos y capas haya, pero su escalado es lineal en N (dadas ciertas arquitecturas). Han demostrado poder aprovechar cantidades enormes de datos (por ejemplo, modelos de lenguaje se entrenan con cientos de gigabytes de texto) siempre que se disponga del cómputo necesario. De hecho, uno de los avances de la última década es que el rendimiento de redes profundas sigue creciendo con más datos y parámetros, a diferencia de otros algoritmos que se saturan.

4. Casos de Uso Típicos

Las redes neuronales artificiales se utilizan hoy en prácticamente todos los ámbitos de la inteligencia artificial. Algunos de sus **casos de uso más importantes** incluyen:

- **Visión Artificial (Computer Vision):** Redes neuronales convolucionales (CNN) y otras arquitecturas visuales han revolucionado la visión por computador. Se emplean en clasificación de imágenes, detección de objetos en tiempo real, segmentación semántica, reconocimiento facial, y muchas aplicaciones médicas. La visión artificial basada en RNA abarca desde filtros de Instagram que detectan facciones, hasta sistemas de vigilancia que buscan actividades sospechosas, o diagnósticos médicos asistidos por IA.

- **Reconocimiento de voz y audio:** Las redes neuronales (especialmente modelos secuenciales como LSTM, y más recientemente transformers) se usan extensamente en reconocimiento de voz: convertir audio hablado a texto (p. ej., los asistentes virtuales tipo Siri, Alexa utilizan RNAs para entender comandos de voz).

- **Procesamiento del Lenguaje Natural (NLP):** Aquí las redes neuronales recurrentes y los transformers han supuesto un salto enorme. Se usan en traducción automática, análisis de sentimientos y clasificación de textos, chatbots y asistentes conversacionales, resumen automático de documentos, extracción de información (encontrar entidades y relaciones en textos), etc.

- **Motores de recomendación:** Grandes plataformas (Netflix, Amazon, YouTube, Spotify) emplean redes neuronales para recomendar contenidos personalizados. Analizan el comportamiento previo del usuario (historial de vistas, compras, gustos) y con redes profundas aprenden a predecir qué producto, película o canción puede interesarle.

- **Generación de contenido (IA generativa):** Las variational autoencoders, GANs (Generative Adversarial Networks) y más recientemente los diffusion models son todos basados en redes neuronales que generan datos nuevos: imágenes sintéticas fotorrealistas, voces imitadas, piezas musicales, etc. Por ejemplo, StyleGAN (red generativa) crea rostros humanos inexistentes con calidad fotográfica. Estas redes han abierto aplicaciones creativas y también controversiales (deepfakes). En texto, modelos como GPT crean contenido escrito. En imágenes, Stable Diffusion genera arte a partir de descripciones.

5. Aplicabilidad a su Proyecto

Para el objetivo de extraer información estructurada de facturas vehiculares (documentos no estructurados, imágenes escaneadas) mediante OCR, las redes neuronales son **pieza central** de la solución propuesta:

- **OCR basado en redes neuronales:** Los sistemas OCR modernos utilizan redes neuronales profundas, típicamente combinaciones de CNN + RNN/LSTM o modelos tipo transformer, para reconocer texto dentro de imágenes. Por ejemplo, el motor OCR open-source Tesseract en sus versiones recientes emplea una LSTM (red neuronal recurrente) para secuenciar caracteres reconocidos. En nuestro proyecto, una red neuronal convolucional puede actuar como extractor de características visuales del documento (detectando dónde hay texto, normalizando la imagen) y luego una red recurrente puede tomar esas características y secuenciarlas para generar el texto reconocido carácter por carácter.

- **Detección de campos en el documento:** Más allá del simple reconocimiento de texto, el proyecto requiere estructurar la información (campos clave como número de placa, fecha, valor, etc.). Aquí también las redes neuronales pueden ayudar mediante modelos de segmentación o etiqueta de secuencias. Un enfoque viable es usar un modelo basado en transformers diseñado para documentos, como LayoutLM, que es una red neuronal pre-entrenada que combina texto + posición para entender documentos. LayoutLM y similares pueden, dados los tokens OCR con sus coordenadas, etiquetar cada fragmento con la clase de campo correspondiente (por ejemplo, podría marcar "ABC-1234" como Placa y una cadena numérica como Monto).

- **Clasificación de documentos o detección de anomalías:** Si se quisiera primero clasificar el tipo de factura (distintas plantillas según la entidad emisora, etc.), se podría usar una CNN a nivel documento (tomando la imagen completa) para clasificarla. Por ejemplo, entrenar una red que con solo mirar el layout o el logo decida si es factura tipo A o B, lo que luego guiaría qué campos buscar. Las CNN son excelentes extrayendo esas características visuales (logos, formatos) que diferencian documentos.

- **Ventajas en este proyecto:** El uso de redes neuronales permite evitar programar manualmente las miles de variaciones posibles de facturas. En cambio, se le mostrarían suficientes ejemplos a la red y esta aprendería a localizar y leer la información relevante. Dado que es probable que las facturas vehiculares tengan campos similares pero dispuestos de forma variable (según emisión municipal, etc.), una red puede generalizar a nuevos formatos mejor que reglas estáticas. Además, una vez entrenada, la red puede procesar documentos rápidamente y con alta precisión, liberando de trabajo manual de digitación.

- **Viabilidad y recursos:** Entrenar desde cero un modelo neuronal para OCR de facturas requeriría un conjunto de datos con suficientes ejemplos de documentos y sus anotaciones (texto transcrito y campos marcados). Si se dispone de, digamos, unos miles de facturas escaneadas con sus datos estructurados correspondientes, se puede lograr. Posiblemente se aprovecharían modelos pre-entrenados (por ejemplo, usar Tesseract o un modelo de reconocimiento de texto preentrenado y luego afinarlo a las facturas locales). El cómputo requerido podría ser alto (entrenar CNN/transformer en GPU). No obstante, existen implementaciones open-source y frameworks como LayoutLM donde solo habría que fine-tunear el modelo con un conjunto más pequeño, lo que sí es abordable con una GPU en tiempos razonables.

Transformers

1. Descripción Teórica

Los Transformers son una arquitectura de red neuronal profunda introducida en 2017, diseñada originalmente para procesar secuencias de datos (especialmente texto) mediante mecanismos de auto-atención masiva en paralelo. A diferencia de las redes recurrentes (RNN) que procesaban las secuencias paso a paso, un Transformer evalúa todas las posiciones de la secuencia simultáneamente, usando mecanismos de atención que le permiten ponderar la importancia relativa de cada elemento en relación con los demás. En otras palabras, la capa central de un Transformer es la self-attention: para cada palabra (o token) de entrada, calcula pesos de atención respecto a todas las otras palabras de la

secuencia, y genera una combinación ponderada que representa el contexto relevante para esa palabra. Esto le permite capturar dependencias de largo alcance de manera efectiva (una palabra al inicio puede influir fuertemente en otra al final, si el peso de atención así lo indica) sin necesidad de recorrer la secuencia secuencialmente.

El Transformer está compuesto por bloques repetitivos llamados bloques de transformación, que incluyen una subcapa de multi-head self-attention y una subcapa de feed-forward (MLP) posterior. El multi-head attention realiza varios mecanismos de atención en paralelo (heads) con distintas proyecciones lineales, permitiendo a cada cabeza atender a diferentes aspectos (por ejemplo, una cabeza podría enfocarse en relaciones sintácticas mientras otra en asociaciones semánticas). Matemáticamente, la atención se computa a través de matrices Q (queries), K (keys) y V (values) obtenidas de la entrada: los pesos de atención son proporcionales a softmax(QK^T) , y estos se usan para combinar los values. Al tener múltiples cabezas, se concatenan sus resultados y pasan por la MLP. Además, los Transformers usan conexiones residuales y normalización por capas para facilitar el flujo de gradiente en redes profundas.

Los Transformers realizan aprendizaje supervisado o auto-supervisado: por ejemplo, BERT se entrena auto-supervisadamente ocultando palabras y pidiendo a la red que las prediga (modelo de lenguaje enmascarado), mientras GPT se entrena a predecir la siguiente palabra (modelo de lenguaje causal). Luego, se fine-tunean (supervisadamente) en tareas específicas. Sus hiperparámetros principales incluyen: el número de capas de atención (depth), el número de cabezas de atención en cada capa, la dimensión de los vectores de embedding (d_model) y la dimensión de la subcapa feed-forward, el tipo de positional encoding utilizado, tasas de dropout, etc. Por ejemplo, el Transformer base tenía 6 capas en encoder y 6 en decoder, con 8 cabezas cada una y dimensión d_model=512. Escalar estos parámetros produce modelos más potentes pero exponencialmente más costosos de entrenar.

2. Ventajas y Desventajas

Ventajas:

- **Altamente paralelizables:** A diferencia de modelos secuenciales (RNN) que procesan token por token, los Transformers procesan distintas partes de la secuencia simultáneamente gracias al mecanismo de atención en paralelo. Esto significa que en hardware GPU/TPU se pueden calcular atenciones para todos los pares en un solo paso matricial. En entrenamiento, esta paralelización acelera significativamente el tiempo necesario para aprender sobre secuencias largas.
- **Captura dependencias de largo plazo eficazmente:** Gracias a la auto-atención, los Transformers son muy buenos modelando contextos amplios. Pueden aprender que la primera palabra de un párrafo influye en la última, o que una referencia en página siguiente conecta con algo anterior,

porque no sufren el problema de vanishing gradient por distancia temporal como las RNN. Esto les permite comprender mejor el contexto global de oraciones y documentos, produciendo texto más coherente y respuestas más contextualizadas. Tareas como resumen de documentos largos o respuestas a preguntas con párrafos contextuales se benefician de esta capacidad de mirar "de un vistazo" toda la entrada.

- **Flexibilidad y adaptabilidad:** La arquitectura Transformer es general y ha sido adaptada a múltiples dominios. No está limitada a texto: variantes han incorporado entradas visuales (Transformers en visión que tratan pixeles como secuencia, o imágenes divididas en parches) y entradas de audio. Esto los hace muy flexibles: el mismo bloque de atención puede utilizarse en diferentes modalidades con mínimos cambios. Además, son escalables: simplemente aumentando capas y cabezas, se ha visto que el desempeño sigue mejorando (hasta cierto punto), lo que ha dado pie a modelos cada vez más grandes y potentes. Los Transformers han dominado NLP (traducción, lenguaje generativo) y están incursionando en visión y otros campos de manera unificada.
- **Estado del arte en múltiples tareas:** Actualmente, los modelos basados en Transformers (BERT, GPT, T5, RoBERTa, etc.) logran resultados líderes o cercanos al mejor en traducción automática, respuesta a preguntas, summarización, generación de código, juegos de lenguaje, e incluso han sido aplicados a secuenciación de proteínas, resolución de problemas matemáticos, etc. Son la base de los grandes modelos de lenguaje (LLMs) como GPT-5, que han demostrado capacidades asombrosas en entender y generar lenguaje natural. La ventaja es que un único marco arquitectónico ha logrado unificar soluciones a problemas muy diversos antes tratados con enfoques específicos.

Desventajas:

- **Alta demanda computacional (especialmente con secuencias largas):** La mayor debilidad práctica de los Transformers es que la auto-atención tiene complejidad cuadrática en la longitud de la secuencia. Si la entrada tiene n tokens, calcular atenciones entre todos los pares cuesta $O(n^2)$ en tiempo y memoria para cada capa. Para secuencias largas, esto se vuelve prohibitivamente costoso.
- **Necesidad de datos masivos y entrenamiento largo**: Los Transformers suelen tener muchísimos parámetros (BERT base ~110M, GPT-3 175B) y por tanto requieren conjuntos de datos muy grandes para entrenarse adecuadamente y evitar sobreajuste. Por ejemplo, GPT-3 se entrenó con prácticamente todo Internet textual. Esto implica que entrenar un buen Transformer desde cero está fuera del alcance de muchos proyectos pequeños, teniendo que recurrir a modelos preentrenados. Incluso con datos, el tiempo de entrenamiento es largo (días o semanas en hardware especializado).
- **Complejidad técnica y ajustes finos:** Aunque la arquitectura es elegante, implementar y entrenar Transformers correctamente requiere manejar muchos detalles. Son modelos complejos con posibles fallos (p. ej., divergencia si la tasa de aprendizaje no es adecuada, inestabilidad de

entrenamiento en redes muy profundas). Además, para tareas específicas casi siempre se necesita fine-tuning y ajuste de hiperparámetros. Todo esto los hace más difíciles de entrenar correctamente sin expertise, comparado con modelos más simples.

- **Tendencia a alucinaciones y falta de constricción:** En modelos generativos, se ha notado que los Transformers (como ChatGPT, etc.) pueden generar contenido plausible pero incorrecto, alucinando datos inexistentes. Esto proviene en parte de la forma en que aprenden patrones estadísticos sin anclaje a la realidad. Si bien esto es un detalle de entrenamiento/datos más que de la arquitectura en sí, es una desventaja en aplicaciones donde la veracidad es crucial. También son menos interpretable incluso que una RNN en el sentido de que aunque se pueden inspeccionar pesos de atención (lo que da cierta explicabilidad), no es trivial extraer de ahí una lógica clara de decisión.

3. Complejidad Computacional

Como se mencionó, la complejidad temporal de los Transformers está dominada por la operación de auto-atención. Para una secuencia de longitud n y dimensión de modelo d, cada cabeza de atención calcula similitudes entre todos los pares de tokens: eso implica operaciones del orden $O(n^2 \cdot d)$. Con h cabezas y L capas, la complejidad por forward-pass es $O(L \cdot h \cdot n^2 \cdot d)$. Habitualmente se considera $h \cdot d$ como aproximadamente constante (p. ej., en Transformers base $d = 512, h = 8$, así que cada capa es $O(n^2 * 512)$ ≈ O(n^2) de orden constante 512). Por tanto, simplificando, atención es $O(n^2)$ en tiempo. La parte feed-forward de cada capa es lineal en n (aplica una misma MLP a cada token, costo $O(n * d * d_{ff})$ para la subcapa densa). Para n grande, la atención domina. Así, duplicar la longitud de secuencia cuadruplica el costo, lo que es bastante severo. Esto hace que entrenar con n=1024 en vez de 512 sea 4 veces más lento, por ejemplo.

La complejidad espacial también escala cuadráticamente con n: se almacena una matriz de atención de $n \times n$ por cada cabeza (aunque no siempre explícitamente, pero sí gradientes de ese tamaño). En términos de parámetros, un Transformer tiene parámetros $O(d^2 L)$ aproximadamente (principalmente en las matrices de proyección de atención y en los MLP); esto suele ser menos problemático comparado con la memoria usada por activaciones si n es grande. La memoria total durante entrenamiento es alta porque se guardan los activaciones de cada capa para backprop, lo que incluye las matrices de atención de tamaño n^2 . Por eso, entrenar modelos con secuencia larga requiere GPUs con mucha VRAM o distribuir en varias.

Para predicción/inferencia, la complejidad por paso sigue siendo O(n^2) para procesar toda la secuencia de golpe. En tareas de generación secuencial (como autocompletar texto con un decodificador), se pueden optimizar calculando incrementamente la atención (almacenando K y V de estados previos para no recomputar) y así cada nuevo token generado cuesta O(n * d) en vez de

recomputar toda la secuencia desde scratch. No obstante, en batch general, procesar secuencias con Transformers es más costoso que con RNN (O(n) por paso) cuando n es muy grande.

Escalabilidad: Los Transformers escalan bien con el tamaño del modelo (parámetros) en el sentido de aprovechar hardware paralelo; de hecho, la investigación ha mostrado que entrenar modelos más grandes con más datos suele ser más eficiente que entrenar modelos pequeños con menos datos para ciertos niveles (Ley de escalamiento). Sin embargo, escalar en longitud de secuencia sigue siendo el principal cuello de botella. Se han propuesto Transformers eficientes para mitigar esto: por ejemplo Sparse Attention (atender solo a algunos tokens, reduciendo cálculo), Linformer (aproxima la matriz de atención reduciendo dimensionalidad), Reformer (usa hashing para aproximar atención), etc. Estos métodos reducen la complejidad a subcuadrática en ciertos supuestos, pero pueden perder algo de precisión. En aplicaciones donde n no es extremadamente grande (chatbots suelen limitar contexto a unos miles de tokens), un modelo transformer normal es manejable con buen hardware.

4. Casos de Uso Típicos

Los Transformers han encontrado usos en una amplia gama de tareas, destacando en Procesamiento de Lenguaje Natural y expandiéndose a otras áreas:

- **Traducción automática y procesamiento de lenguaje:** El Transformer original fue concebido para traducción y actualmente casi todos los sistemas de traducción de vanguardia (Google, DeepL, etc.) usan Transformers entrenados en corpus bilingües enormes. Además, modelos como BERT (codificador) se usan en análisis de texto para tareas de clasificación (sentimiento, intentos), reconocimiento de entidades, respuesta a preguntas. Los Transformers generativos (GPT, T5) se emplean en resumen de documentos, generación de texto (redacción asistida, chatbots avanzados), extracción de información y relleno de espacios en texto. ChatGPT, por ejemplo, puede considerarse un Transformer decodificador gigantesco entrenado con afinado conversacional.

- **Visión por Computador:** Aunque inicialmente el dominio de visión era territorio de las CNN, recientemente los Vision Transformers (ViT) y variantes han demostrado rendimientos competitivos e incluso superiores en clasificación de imágenes, detección de objetos y segmentación. Un Vision Transformer trata una imagen dividida en parches como una secuencia de tokens visuales, y aplica atención para determinar relaciones espaciales/patronales. En OCR, Transformers se usan para leer texto en imágenes (como TR-OCR o Donut model) directamente.

- **Grandes Modelos de Lenguaje (LLMs) y asistentes conversacionales:** Uno de los casos más notables es la creación de modelos como GPT-5, Grok, Gemini, etc., que son Transformers gigantes entrenados con cantidades enormes de texto. Estos modelos pueden hacer desde resolver problemas de programación hasta componer ensayos, traducir, responder preguntas complejas e incluso aprobar exámenes. Han dado pie a aplicaciones comerciales como ChatGPT, Grok, Gemini de Google.

5. Aplicabilidad a su Proyecto

En el problema de extraer información de facturas vehiculares, los Transformers pueden aportar capacidades avanzadas para entender el contexto textual y visual del documento:

- **Modelo de comprensión de documentos:** Una de las soluciones más poderosas actualmente para extracción de campos de documentos es usar un modelo Transformer especializado como LayoutLM. LayoutLM es básicamente un Transformer entrenado para combinar la información de texto (palabras reconocidas) con su posición en la página (codificada como embeddings espaciales). Al pasar todos los tokens de la factura a este modelo, puede etiquetar cada token con categorías como "Nombre del propietario", "Número de placa", "Monto", etc. Esto se ajusta perfectamente a nuestro objetivo de obtener información estructurada. Con un fine-tuning adecuado en facturas vehiculares (aportando algunas docenas o cientos de facturas etiquetadas con sus campos), el Transformer aprendería las regularidades de formato (por ejemplo, que una palabra seguida de "Placa:" suele ser la placa del auto, o que un valor con "$" en cierto sector es el monto). La ventaja es que captura relaciones globales: si el formato de una factura cambia de orden de campos, el modelo aún podría identificar cada campo atendiendo al texto circundante y posición, algo más difícil de lograr con reglas fijas.

- **Mejor manejo de variabilidad de formato**: Las facturas de distintos municipios o entes pueden variar en estructura (algunas ponen el monto arriba, otras abajo, algunos usan palabras clave distintas como "Matricula" vs "Placa"). Un Transformer entrenado en datos variados podrá generalizar a nuevas variaciones atendiendo tanto a las palabras clave presentes como a su contexto. Por ejemplo, si nunca vio "Reg. No." pero sí "Registro:", podría inferir por contexto que "Reg. No." seguido de una cadena alfanumérica es similar a "Número de Registro". Esta capacidad de entender sinónimos y contexto lo da el pre-entrenamiento en lenguaje amplio (si partimos de LayoutLM pre-entrenado en millones de documentos).

- **Integración con OCR:** Podríamos plantear un pipeline donde primero un OCR extrae los textos con sus coordenadas, y luego un Transformer (ej. LayoutLM) toma esos textos posicionados y los estructura. Alternativamente, hay investigaciones de modelos end-to-end (como Donut de NAVER) que son transformers puros end-to-end capaces de tomar la imagen del documento directamente como entrada y generar como salida un JSON con los campos. Donut evita el OCR explícito; usa un Vision Transformer para extraer visual features y un Transformer decodificador para producir la información estructurada. Esta clase de modelos es muy nueva pero promete simplificar la arquitectura. En cualquier caso, los Transformers permiten un **paso de comprensión** mucho más profundo que métodos previos.

- **Clasificación de tipo de documento o detección de fraude con contexto:** Además de la extracción de campos, un Transformer podría entrenarse para clasificar la factura (por ejemplo, si hubiera diferentes tipos de formularios). Dado el texto completo, un encoder Transformer podría aprender representaciones diferenciadas para, digamos, facturas de compra-venta vs facturas de matriculación, etc., si eso fuera relevante. Igualmente, para fraude detection, un Transformer podría ser entrenado para detectar anomalías en el texto completo (como datos incoherentes). Por ejemplo, si una factura vehicular tiene un formato muy distinto a los vistos o valores ilógicos, el modelo podría detectarlo por no encajar en las atenciones esperadas.

- **Inclusión en el proyecto:** Transformers encajan de forma natural en un proyecto de OCR inteligente. Nuestra tarea combina visión (leer texto) y lenguaje (interpretar qué significa ese texto en contexto de un documento), y los Transformers son actualmente la mejor herramienta para tareas de Vision+Language. Incluir un componente basado en Transformers (sea un modelo pre-entrenado adaptado o uno entrenado específicamente) probablemente incrementará la robustez ante distintas plantillas y reducirá los errores de extracción. La alternativa sería programar reglas manuales para cada variación de factura, lo cual es menos escalable. Por tanto, la aplicabilidad al proyecto es muy alta: un Transformer bien entrenado puede servir de "cerebro" que entiende la factura una vez que el OCR le provee el texto, dándole al sistema la capacidad de generalizar y mejorar conforme vea más datos (incluso se podría entrenar continuamente con nuevas facturas verificadas, aprovechando el aprendizaje profundo). En resumen, invertir en esta técnica dentro del proyecto aporta modernidad y potencial de alcanzar alta precisión en la tarea de extracción de información estructurada.

Redes Neuronales Convolucionales (CNN)

1. Descripción Teórica

Las Redes Neuronales Convolucionales (CNN) son un tipo especializado de red neuronal diseñada para procesar datos con estructura de grid o rejilla, como imágenes (2D) o series temporales (1D), aprovechando la correlación espacial/local presente en ellos. En una CNN, en lugar de conectividad densa entre capas (todas las neuronas conectadas con todas), se usan conexiones locales: cada neurona en una capa convolucional recibe entrada solo de una región pequeña de la capa anterior (llamada campo receptivo), y múltiples neuronas (filtros) exploran toda la entrada mediante convolución con pesos compartidos.

El principio matemático es la operación de convolución discreta: se define un kernel (filtro) una pequeña matriz de pesos, por ejemplo de tamaño 3x3 - y se desplaza (desliza) sobre la entrada (imagen) calculando productos puntuales y sumas. Cada filtro detecta cierto tipo de característica local (por ejemplo, un borde horizontal, una textura particular). Los pesos de estos filtros se aprenden durante el entrenamiento. Al aplicar un filtro sobre toda la imagen, se genera un mapa de características que resalta dónde aparece ese patrón en la imagen. Una capa convolucional típicamente tiene múltiples filtros aprendidos (decenas o cientos), por lo que produce múltiples mapas de características en paralelo.

Adicionalmente, las CNN usan capas de agrupamiento (pooling), que reducen la resolución espacial de los mapas (por ejemplo, tomando el máximo o promedio en regiones 2x2). Esto introduce invarianza a pequeñas traslaciones: pequeñas variaciones de posición en la característica no cambian la activación máxima. También reduce la dimensionalidad, resumiendo la información. Tras varias capas de convolución + pooling, usualmente se agregan capas totalmente conectadas finales para tomar las características extraídas y producir la salida (por ejemplo, probabilidades de clases).

Los fundamentos matemáticos se basan en álgebra lineal (multiplicación de matrices grandes si se desenrolla la convolución) y en particular la operación de correlación. Comparado con una red densa, una CNN tiene muchos menos parámetros porque cada filtro es pequeño y compartido en toda la entrada (por ejemplo, un filtro 3x3 tiene 9 pesos que se aplican en cada posición, en lugar de pesos individuales para cada pixel). Esto actúa como regularización incorporada: la suposición de localidad y compartición es adecuada para imágenes, donde un patrón puede aparecer en cualquier lugar.

El tipo de aprendizaje sigue siendo supervisado en la mayoría de casos (p.ej. etiquetamos imágenes y la CNN aprende a extraer características relevantes para clasificar). También se emplean CNN en

aprendizaje no supervisado (como en autoencoders convolucionales que detectan patrones sin etiquetas). Los hiperparámetros principales incluyen: el número de filtros por capa (y sus tamaños), el tamaño del filtro (ej. 3x3, 5x5), el stride o paso de desplazamiento del filtro, si se usan padding (relleno) para conservar tamaños, la arquitectura en sí (cuántas capas conv, dónde aplicar pooling, etc.), además de los habituales de entrenamiento (learning rate, etc.). Las funciones de activación en CNN suelen ser ReLU para inducir no linealidad después de cada convolución.

2. Ventajas y Desventajas

Ventajas:

- Especializadas en datos visuales/espaciales: Las CNN logran rendimiento superior en visión artificial comparado con redes densas u otros métodos, ya que explotan la estructura local de las imágenes. Detectan eficientemente bordes, texturas y composiciones jerárquicas de estas. Antes de las CNN, la extracción de características visuales era manual y lenta; las CNN introdujeron un enfoque escalable y automático para reconocimiento de patrones visuales.

- Menos parámetros y más eficiencia que redes densas: Gracias a la compartición de pesos, una CNN típica tiene muchos menos parámetros que una red totalmente conectada de tamaño equivalente. Por ejemplo, una imagen de 100x100 (~10k inputs) conectada densamente a 100 neuronas serían ~1e6 pesos, mientras una conv de 10 filtros 5x5 tendría solo 250 pesos por filtro = 2500 pesos, pero cubre toda la imagen.

- Invariancia a traslaciones y distorsiones pequeñas: Mediante pooling y el hecho de aplicar el mismo filtro en toda la imagen, las CNN toleran que un patrón aparezca en distintas posiciones. Un detector de borde horizontal reaccionará igual esté el borde arriba o abajo en la foto. Esto otorga robustez a desplazamientos, escalados pequeños y ruido. Modelos antiguos necesitaban generar features para cada posición; la CNN lo resuelve en uno solo.

Desventajas:

- Altamente demandantes en cómputo (especialmente en entrenamiento): Las CNN, aunque más eficientes que densas, aún requieren una gran cantidad de operaciones, especialmente en capas con muchos filtros y en imágenes de alta resolución. Por ejemplo, procesar una imagen HD con múltiples capas conv y decenas de filtros supone millones de multiplicaciones. Entrenar CNN profundas en CPU es extremadamente lento; típicamente se necesitan GPUs.

- Necesidad de grandes volúmenes de datos etiquetados: Las CNN, como otros modelos de deep learning, suelen requerir bastantes datos de entrenamiento para generalizar bien. Si se intenta entrenar una CNN desde cero con pocas imágenes, es probable que sobreajuste o no aprenda rasgos significativos. Por ello, en visión es común recurrir a modelos pre-entrenados (transfer learning) si el dataset propio es pequeño.

- No son fácilmente interpretables: Aunque un poco más interpretables que una red densa (uno puede visualizar filtros para intuir qué detectan, p. ej. un filtro aprende a detectar "textura de rueda"), en general siguen siendo cajas negras en términos de decisión final.

3. Complejidad Computacional

En términos de big-O, la complejidad temporal de una capa convolucional depende del tamaño de la entrada, el tamaño del filtro y el número de filtros. Supongamos una capa que toma como entrada una imagen/tensor de tamaño $H \times W \times C_{in}$ (alto, ancho, canales de entrada) y tiene C_{out} filtros de tamaño $K_h \times K_w$ cada uno. Para cada posición donde se aplica el filtro, se realizan $K_h \cdot K_w \cdot C_{in}$ multiplicaciones acumuladas para producir una salida. El número de posiciones de aplicación es aproximadamente $H \times W$ (si stride=1 y padding completo, la salida es similar tamaño que entrada). Por tanto, el costo de una capa conv es $O(H \cdot W \cdot C_{in} \cdot C_{out} \cdot K_h \cdot K_w)$. Si usamos stride > 1 o pooling, H y W se reducen en siguientes capas, pero grosso modo para calcular la complejidad total de la red sumamos esta expresión para cada capa. Por ejemplo, una conv 3x3 (K=3) en una imagen 224x224 con 64 filtros y 3 canales de entrada tiene $224 * 224 * 3 * 64 * 9 \approx 87 \times 10^6$ operaciones, lo cual es bastante. Una red como VGG16 realiza varias de esas (por eso VGG16 tenía del orden de 15.5 billones de FLOPs).

La complejidad espacial en términos de memoria durante entrenamiento incluye almacenar los pesos de los filtros (lo cual es pequeño comparado a densas: por ej. 64 filtros 3x3x3 son 1728 pesos), pero sobre todo las activaciones intermedias (feature maps). Esas pueden ser grandes: por ej. primera capa de ResNet50 produce 64 mapas de 112x112, que son ~0.8 million values; en capas profundas a veces el número de canales es grande (256x14x14, etc.). Durante backprop, hay que almacenar también gradientes de activaciones de tamaño similar. Por eso, CNN profundas consumen mucha VRAM. Sin embargo, con técnicas como half precision, memory reuse, etc., se maneja.

Considerando predicción/inferencia, la complejidad es similar a forward (sin backprop).

Convoluciones se pueden optimizar usando transformadas (FFT) o im2col + GEMM para aprovechar hardware. En general, la inferencia de una CNN se mide en FLOPs (ej. X FLOPs por imagen). Redes pesadas pueden tardar decenas de milisegundos por imagen en GPU, o más en CPU. Se han diseñado variantes más eficientes (MobileNet, SqueezeNet) para dispositivos móviles, reduciendo computación con depthwise separable conv.

Escalabilidad: Entrenar CNN más grandes linealmente aumenta cómputo con más filtros/capas, pero se puede paralelizar el cálculo de diferentes filtros y diferentes imágenes en un batch bastante bien en GPUs. Las CNN escalan con datos: con más imágenes se reducen error hasta saturar. Hay un límite práctico: redes extremadamente grandes sufren de sobreajuste si no hay datos suficientes, por lo que la escalabilidad se equilibra con técnicas de regularización y dataset. A nivel implementativo,

distribuir una CNN en múltiples GPUs es factible repartiendo el batch, porque las operaciones son locales. Algunas CNN gigantes (Ej: modelos vision transformer híbridos) han usado múltiples dispositivos.

4. Casos de Uso Típicos

Las redes convolucionales son omnipresentes en aplicaciones de visión por computador y también se utilizan en otros dominios. Algunos casos de uso típicos:

- **Clasificación de imágenes:** Identificar el contenido principal de una imagen (ej. qué objeto o escena aparece). Aplicaciones: sistemas de etiquetado de fotos (Google Photos reconoce personas, lugares), diagnóstico médico por imágenes (clasificar radiografías en sano vs enfermo), sistemas de vigilancia (detectar si una cámara ve a una persona, vehículo, etc.).

- **Detección de objetos:** Localizar y clasificar múltiples objetos dentro de una imagen (salida: cajas delimitadoras y clases). Esto es crucial en vehículos autónomos (detectar peatones, señales), seguridad (detectar intrusos en cámaras), análisis de tráfico (contar coches, identificar matrículas), etc. Modelos típicos: YOLO, SSD, Faster R-CNN (muchos de los cuales usan CNN base para extraer características). Por ejemplo, YOLO puede en tiempo real detectar decenas de objetos por frame de video gracias a CNN optimizadas.

- **Reconocimiento facial:** Identificar rostros y/o verificar identidad. CNN se utilizan para face detection (encontrar caras en una imagen) y luego para face recognition (comparar con una base de datos). Aplicaciones: desbloqueo facial de smartphones, etiquetado automático en redes sociales, sistemas de control de acceso, vigilancia. Redes como FaceNet (Google) o DeepFace (Facebook) lograron representar caras en vectores con enorme precisión usando CNN profundas entrenadas en millones de rostros.

- **Procesamiento de texto (NLP) sencillo:** Antes del auge transformer, CNN 1D se usaban para clasificación de textos cortos (como sentiment analysis) extrayendo n-grams importantes. Por ejemplo, una CNN con filtros de ancho 3-5 podía detectar frases como "muy bueno" o "no recomendada" para calificar reseñas. Aún son útiles en tareas con textos concisos como categorización de tweets, ya que son más rápidas que RNN.

5. Aplicabilidad a su Proyecto

En el proyecto de extracción de información de facturas vehiculares mediante OCR, las CNN tienen un rol fundamental en la etapa de procesamiento de imagen para reconocer texto:

- **Motor OCR basado en CNN:** El reconocimiento óptico de caracteres moderno suele apoyarse en CNN para identificar letras o palabras en segmentos de imagen. Por ejemplo, una arquitectura típica es usar una CNN para extraer características de la imagen del texto, seguida de una RNN o CTC decoder para secuenciar los caracteres. La CNN podría ser entrenada para producir un mapa de probabilidad por cada posible carácter en pequeñas ventanas deslizantes. De hecho, muchas soluciones OCR open-source (Tesseract 4, por ejemplo) incorporan una CNN (en Tesseract es parte de la "LSTM engine", hay una capa convolucional inicial). En nuestro proyecto, una CNN bien entrenada en dígitos/letras de placas o en tipografías de facturas mejorará la precisión de lectura comparado con métodos más antiguos basados en correlación de plantillas. Sin la CNN, el sistema de OCR sería mucho menos robusto ante variaciones de fuente, ruido, baja resolución, etc.

- **Detección de áreas de texto en la factura:** Antes de leer, a veces es útil detectar dónde hay texto en el documento (especialmente si el documento es semiestructurado con logos, tablas, sellos). Una CNN especializada (por ejemplo, usando un modelo de segmentación) podría localizar bloques de texto o campos específicos (como hallar dónde está el número de placa visualmente buscando un patrón de caracteres alfanuméricos con cierto formato). Existen algoritmos de text detection como EAST o CTPN que usan redes convolucionales para escanear la página y encerrar regiones que probablemente contienen texto. Integrando algo así, podríamos segmentar la factura en regiones (cabecera, cuerpo, pie) que luego se pasen al reconocimiento detallado.

- **Implementación y viabilidad:** Afortunadamente, no es necesario desarrollar una CNN OCR desde cero. Hay librerías y modelos pre-entrenados (como EasyOCR, OCRopy, Tesseract con modelos neuronales) que podríamos utilizar o tomar como base. Probablemente usaríamos transfer learning: por ejemplo, una CNN pre-entrenada en reconocer caracteres latinos, afinándola con ejemplos de nuestras facturas (especialmente para cualquier fuente especial, logos, etc.). El costo computacional de inferencia de la CNN en OCR es razonable; hoy día un smartphone puede ejecutar OCR en tiempo real para una foto.

- **Justificación de uso:** Sin duda, la técnica de CNN es imprescindible en el proyecto porque es la piedra angular para convertir la imagen en texto digitalizado confiable.

- **Relación con otras técnicas:** Cabe mencionar que en este proyecto las CNN trabajan de la mano con otras técnicas ya mencionadas: por ejemplo, la CNN proporciona la secuencia de caracteres (o sus características) que luego un Transformer o una red neuronal secuencial puede tomar para estructurar la info. También podríamos usar Random Forest o SVM para

verificar algún dato específico extraído, pero la CNN es insustituible en la etapa de percepción visual.